K. N. Toosi
University of Technology

Digital Image processing

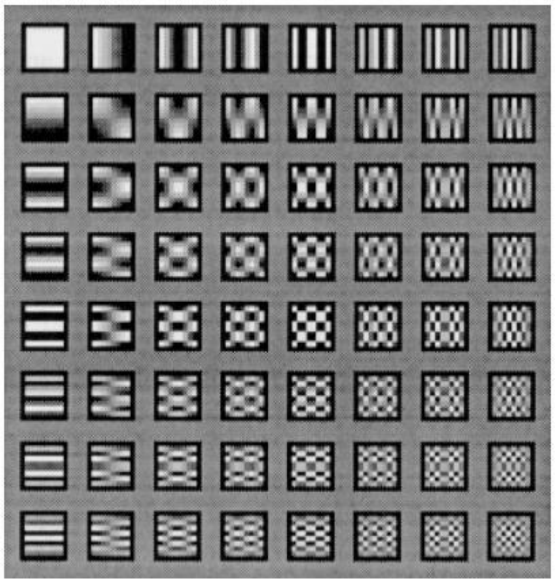
CHW3

Due: 99/2/8

Machine Vision & Image Processing Lab.

JPEG Compression:

In this exercise we plan to implement a simplified version of JPEG compression standard. The algorithm behind the JPEG standard comes from Discrete Cosine Transform. A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. Therefore, an image can be represented by a linear combination of Discrete Cosine Transform basis images as illustrated in the following figure. This figure shows 64 DCT basis images corresponding to an 8x8 image.

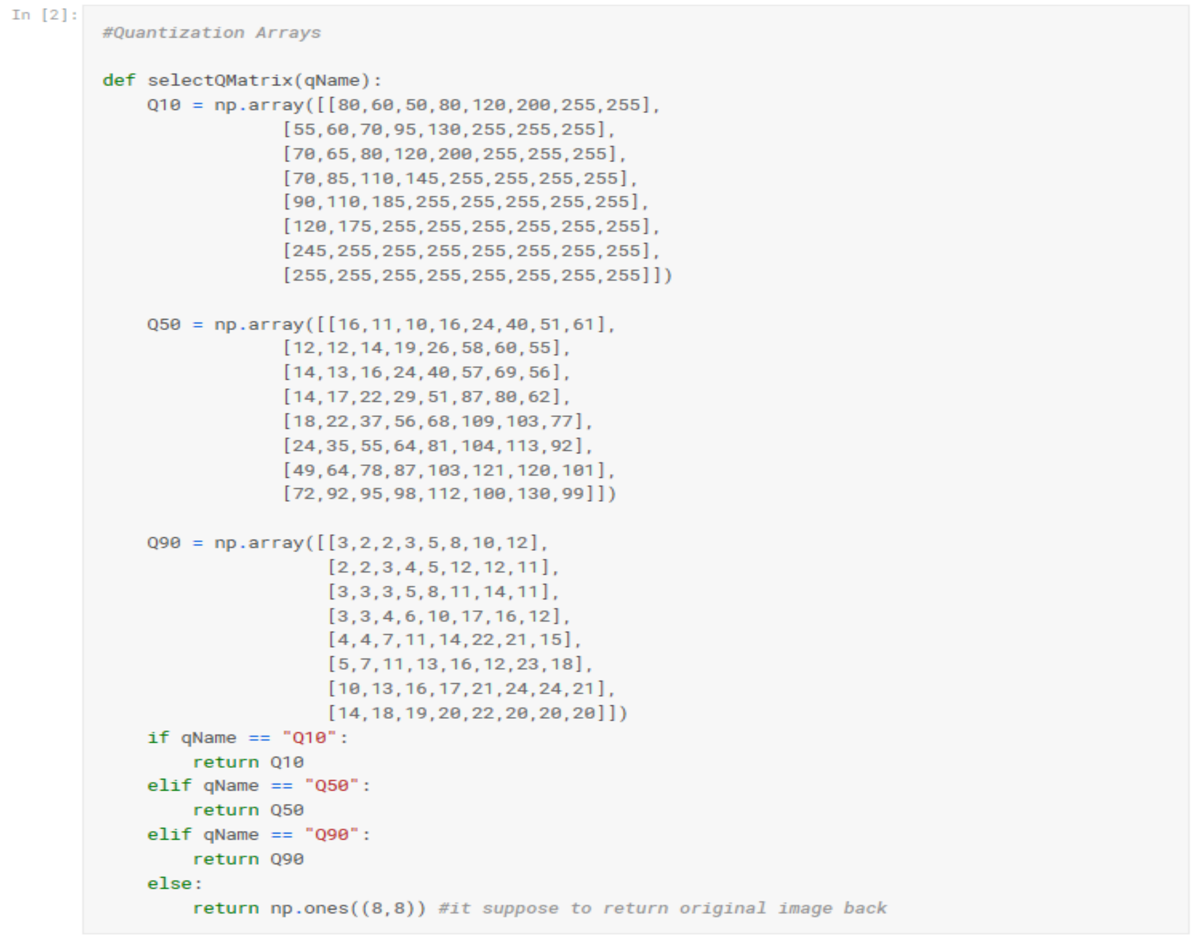


In JPEG standard, the input image is first partitioned into a number of non-overlapping 8×8 blocks. Then, DCT transforms each 8×8 block of input image to a linear combination of these 64 patterns. The patterns are referred to as the two-dimensional DCT basis functions, and the output values are referred to as transform coefficients. As illustrated in the above figure, the basis images placed in the top left part of the figure concern low spatial frequencies while the basis images in bottom right concern high spatial frequencies. In other words, each basis image can be considered as a bandpass spatial filter and its bandwidth is determined by its position in the above figure, such that top left basis images correspond to low frequencies and those in the bottom right correspond to high frequencies. An image actually includes most of these frequencies, and it can be represented with linear combination of these basis images. In the next step, DCT coefficients of each 8×8 block are normalized using a quantization array, elementwise and subsequently quantized. The quantized coefficients are then scanned diagonally starting from the DC coefficient and coded using a variable length coding algorithm.

Reason for Quantization & Quantization Arrays:

The human eye is able to detect small intensity changes in relatively uniform regions of an image, while it ignores large brightness variations in (non-uniform) regions with high intensity variations. This allows one to greatly reduce the amount of data in the high frequency components in DCT coefficients. This is done by simply dividing each component of DCT by a proper constant for that component, and then rounding the result to nearest integer. This rounding operation is the only lossy operation in the whole process if the DCT computation is performed with sufficiently high precision. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and The remaining ones become small positive or negative numbers, which take many fewer bits to be represented. Since DCT coefficients will be divided to quantization arrays elementwise, amplifying the value of quantization array elements will cause higher compression of the image. Consequently, the compressed image size or more precisely, the number of nonzero quantized coefficients (which will be saved on the memory) are expected to reduce as quantization array values augment. That means:

# of nonzero codes resulted by Q10< # of nonzero codes resulted by Q50< # of nonzero codes resulted by Q90



In [2]:

```
#Quantization Arrays

def selectQMatrix(qName):
    Q10 = np.array([[80,60,50,80,120,200,255,255],
                [55,60,70,95,130,255,255,255],
                [70,65,80,120,200,255,255,255],
                [70,85,110,145,255,255,255,255],
                [90,110,185,255,255,255,255,255],
                [120,175,255,255,255,255,255,255],
                [245,255,255,255,255,255,255,255],
                [255,255,255,255,255,255,255,255]])

    Q50 = np.array([[16,11,10,16,24,40,51,61],
                [12,12,14,19,26,58,60,55],
                [14,13,16,24,40,57,69,56],
                [14,17,22,29,51,87,80,62],
                [18,22,37,56,68,109,103,77],
                [24,35,55,64,81,104,113,92],
                [49,64,78,87,103,121,120,101],
                [72,92,95,98,112,100,130,99]])

    Q90 = np.array([[3,2,2,3,5,8,10,12],
                    [2,2,3,4,5,12,12,11],
                    [3,3,3,5,8,11,14,11],
                    [3,3,4,6,10,17,16,12],
                    [4,4,7,11,14,22,21,15],
                    [5,7,11,13,16,12,23,18],
                    [10,13,16,17,21,24,24,21],
                    [14,18,19,20,22,20,20,20]])
    if qName == "Q10":
        return Q10
    elif qName == "Q50":
        return Q50
    elif qName == "Q90":
        return Q90
    else:
        return np.ones((8,8)) #it suppose to return original image back
```

The Original Image:

I called show Image function, which is defined above. You can choose between images which are located in input folder by changing the directory variable.

```
import os
from scipy import misc
path = 'your_file_path'
image= misc.imread(os.path.join(path,'image.bmp'), flatten= 0)
```

Block Splitting:

Block splitting is one of the processes that split the image into smaller blocks of 8×8 or 16×16 dimension. The important thing to understand is why this is done.

The first move is reading image with using OpenCV then, extract height and width of the given image. use those variables for splitting operation. create an empty list for storing the image partitioned into 8x8 blocks. "block" parameter determines the block size of partitioned mage.

```
In [4]:
height  = len(img) #one column of image
width = len(img[0]) # one row of image
sliced = [] # new list for 8x8 sliced image
block = 8
print("The image heigh is " +str(height)+", and image width is "+str(width)+" pixels")
```

```
The image heigh is 600, and image width is 800 pixels
```

Dividing into parts:

Before computing the DCT of the 8×8 block, its values are shifted from a positive range to one centered at zero. For an 8-bit image, each entry in the original block falls in the range [0,255]. The midpoint of the range (in this case, the value 128) is subtracted from each entry to produce a data range that is centered at zero, so that the modified range is [-128,127]. This step reduces the dynamic range requirements in the DCT processing stage that follows.

```
: #dividing 8x8 parts
  ....
```

Discrete Cosine Transform:

cv2.dct performs a forward or inverse discrete Cosine transform of 1D or 2D array.

Parameters of cv2.dct (input floating-point array, output array of the same size and type as source, inverse or forward flag)

Calculate the DCT of blocks and append all blocks into DCT output list.

You have a computationally costly algorithm here. Write a code to quantize each pixel in blocks.

Notice that most of the higher-frequency elements of the sub-block (i.e., those with an x or y spatial frequency greater than 4) are diminished into zero values.

You can rearrange the Quantization matrix by changing selectQMatrix input, variable Q. Matrix options are: "Q10", "Q50" and "Q90".

```
: selectedQMatrix = selectQMatrix("Q10")
  ....
```

Inverse Discrete Cosine Transform:

cv2.idct() calculates inverse Discrete Cosine Transform of 1D or 2D array. Parameters of cv2.idct are The same with cv2.dct. Use this function and compute Inverse Discrete Cosine transform of Blocks. Do not forget to add 128 to the IDCT values. This will bring back the intensities in the range of [0 255].

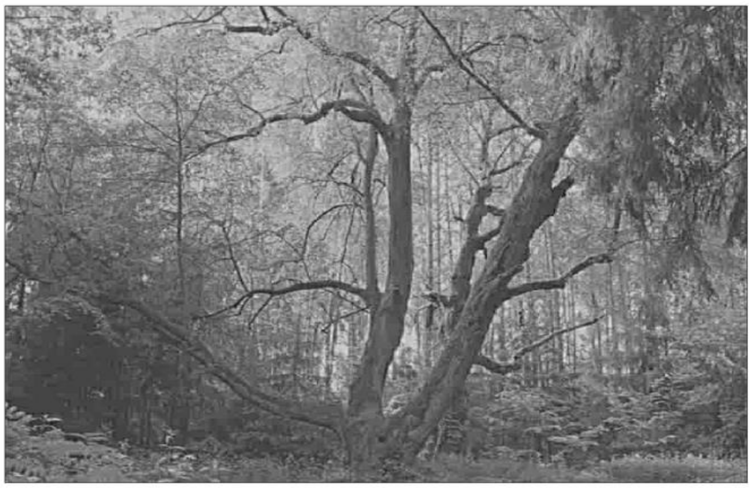
Complete the Puzzle:

Put the Reconstructed blocks together. Consider a list as invList. It is one dimensional list that contains 8x8 numpy arrays inside it. In order to recover the image:

1. invList elements should be parsed at width/block steps. For example: If the image width is 1200 pixels, parsing steps should be 1200/8=150 pixels
2. Use np.hstack if you are going to recover the image's rows first, else use np.vstack for recover columns first.
3. Append all recovered columns into another list.
4. Apply np.vstack if you have been used np.hstack, reverse order with entry 2

Results:

Here it is, the decompressed image. As you can see the image lost some high frequency details and it causes reduced size on disk. There is no change in total pixel count, the JPEG did not reduce the size of the image, it just cancelled out some high frequency details which is normally ignored by the human subjects in the decompressed image.



Please compare the size of original image (# of rows× # of columns× # of bytes/pixel) and the compressed image (nonzero quantized codes).

"To escape criticism: do nothing, say nothing, be nothing."